Improving The Scalability And Efficiency Of K-Medoids By Map Reduce

Mr D Lakshmi Srinivasulu, Mr A Vishnuvardhan Reddy, Dr V S Giridhar Akula

***Abstract*— Day to day the size of data increased enormously. Big Data concerns large-volume, complex, growing data sets with multiple, autonomous sources. With the fast development of networking, data storage, and the data collection capacity. Big Data is now rapidly expanding in all science and engineering domains, including physical, biological and bio-medical sciences. Mining knowledge from the large amount of data is a challengeable task. Map Reduce is a programming model and an associated implementation for processing and generating large data sets. Map reduce is one of the technique to achieve parallelism. *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key . Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. In the perspective of clustering, grouping of similar of objects from big data is a challengeable task .In order to deal with the problem; many researchers try to design different parallel clustering algorithms. In this paper, we propose a parallel K-Medoids clustering algorithm to improve scalability without noise and efficiency based on Map Reduce.**

***Index Terms*— Clustering: K-Medoids, Hadoop (HDFS),Map reduce**

I. INTRODUCTION

From day to day data size has been increased enormously in various formats from various sensor devices which are connected to different applications. The existing algorithms don't have the capability to analyze the large amount of data with different formats. The challenges include in 1.Hardware configuration 2.algorithm design. To face these challenge

Hardware configuration,storing the data in Hadoop Distributed File System(HDFS).

HDFS is a file system designed for storing very large files with streaming data access patterns, running on cluster of commodity hardware[1].The size of large file may be hundreds of mega bytes,giga bytes or tera bytes. The total data is distributed in more number of nodes. Hence instead of single system, more number of systems are used to store huge amount of data.

We can face the challenge "algorithm design" by parallel processing.Effecient parallel algorithms and implementation techniques are the key to meet the scalability and performance.Upto now so many parallel clustering algorithms [2,3,4].All these algorithms having the following drawbacks)They assume all objects can reside in main memory at the same time their parallel systems have provided restricted programming models and used the restrictions to parallelize the computation automatically. Both are prohibitive for very large data sets. So there is a need to develop parallel clustering algorithms. Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodty hardware in a reliable, fault-tolerant manner.

The MapReduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the writablecomparable interface to facilitate sorting by the framework.

Input and Output types of a MapReduce job:
(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output).

Map Reduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Computational processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.

"Map" step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

"Reduce" step: The master node then collects the answers to all the sub-problems and combines them in some way to form

Mr D Lakshmi Srinivasulu, Assistant Professor in CSE Dept,G.Pulla Reddy Engineering College(Autonomous) Kurnool,AP
Mr A Vishnuvardhan Reddy, Assistant Professor in CSE Dept,G.Pulla Reddy Engineering College(Autonomous) Kurnool,AP
Dr V S Giridhar Akula, Professor and Principal in Methodist College of Engineering & Technoloy,Hyderabad,Telangana

the output – the answer to the problem it was originally trying to solve.

In this paper, we adapt *K-MEDOIDS* algorithm [5] in MapReduce framework is implemented by Hadoop to make the clustering method applicable to large scale data. By applying proper <key, value> pairs, the proposed algorithm can be parallel executed effectively. We conduct comprehensive experiments to evaluate the proposed algorithm. The results demonstrate that our algorithm can effectively deal with large scale datasets.

The rest of the paper is organized as follows. In Section 2, we present our parallel *K-Medoid* algorithm based on MapReduce framework. Section 3 shows experimental results and evaluates our parallel algorithm with respect to speedup, and sizeup. Finally,we offer our conclusions in Section 4.

II. PARALLEL K-MEDOID ALGORITHM BASED ON MAPREDUCE:

In this section we propose parallel K-MEDOIDS clustering algorithm based on MapReduce.Initially we give brief overview of K-Medoids ,later we explain the how mapreduce will be integrated with the basic K-Medoids.

2.1 K-Medoids Algorithm:

K-Medoids is a representative of object-based technique. It is a partitioning based on medoid or central objects. The K-Medoids method is more robust than k-means in the presence of noise and outliers because a medoid is less influenced by outliers or other extreme values than mean.

The K-Medoids algorithm is as follows:

Algorithm:K-Medoid algorithm for partitioning based on central object.

Input: K: number of clusters, D:a data set containing n objects.

Output set of k clusters.

Method:

1.arbitarly choose k objects in D as the initial representative objects

2.repeat

{

Assign each remaining object to the cluster with the nearest representative object.

Randomly seslect a non representative object ,O_{random} ;

Compute total cost,S,of swapping representative objet,O_j with O_{random}

If S<0 then swap O_j with O_{random} to form the new set of k representative objects;

3.Until no change;

k-medoid is not sensitive to noisy data and outliers, but it has high computation cost.If the data size becomes too large, the computation cost increases extremely. So to overcome this problem implementing the K-Medoids by map reduce technique.

2.2 Parallel K-Medoids based on Map Reduce:

As the analysis above-Medoids algorithm can be implemented by applying the technique MapReduce.The map function performs the procedure of assigning each sample to the closest medoid while the reduce function performs the procedure of updating the new centers.The cost of network communication will be reduces by using a combiner function by partial combination of the intermediate values with the same key within the same map task.

Map-function : The input dataset is stored on HDFS[5] as a sequence file of <key, value> pairs, each of which represents a record in the dataset. The key is the offset in bytes of this record to the start point of the data file, and the value is a string of the content of this record. The dataset is split and globally broadcast to all mappers. Consequently, the distance computations are parallel executed. For each map task, Parallel K Medoids construct a global variant medoids which is an array containing the information about medoids of the clusters. Given the information, a mapper can compute the closest medoid for each sample. The intermediate values are then composed of two parts: the index of the closest center point and the sample information. The map function is shown in Algorithm 1.

Algorithm 1. map (*key*, *value*)
Input: Sample Values
Output: <*key*', *value*'> pair, where the *key*' is the index of the closest center point and *value*' is a string comprise of sample information
1. Construct the sample *instance* from *value*;
2. *minDis* = *Double.MAX VALUE*;
3. *index* = -1;
4. For i=0 to *centers*.length do
dis= *ComputeDist*(*instance*, data[i]);
If *dis* < *minDis {*
minDis = *dis*;
index = *i*;
}
5. End For
6. Take *index* as *key*';
7. Construct *value*' as a string comprise of the values of different dimensions;
8. output < *key*, *value*> pair;
9. End

Step 2 and Step 3 initialize the auxiliary variable *minDis* and *index* ; Step 4 computes the closest center point from the sample, Step 8 outputs the intermediate results.

Combine-function. After each map task, we apply a combiner to combine the intermediate results of the same map task. Here there is no communication cost because intermediate results are stored in local disk of the host. In the combine function, we partially sum the values of the points assigned to the same cluster. In order to calculate the median value of the objects for each cluster, we should record the number of samples in the same cluster in the same map task. The pseudo code for combine function is shown in Algorithm 2.

Algorithm 2. combine (*key*, C)
Input:Subset of original data,Array C stores the samples assigned to the same cluster.
Output:Partial Clusters
1.repeat
2.{
3.Assign each remaining object to the cluster with the nearest representative object.
4.Randomly seslect a non representative object ,O_{random} ;

5.Compute total cost,S,of swapping representative objet,O_j with O_{random}
6.If S<0 then swap O_j with O_{random} to form the new set of k representative objects;
7.number++
7.}Until no change;

Reduce-function.

The input of the reduce function is the data obtained from the combine function of each host. In reduce function, we can sum all the samples and compute the total number of samples assigned to the same cluster. Therefore, we can get the new medians which are used for next iteration. The reduce function is shown in Algorithm3.

Algorithm 3. reduce (*key*, C)

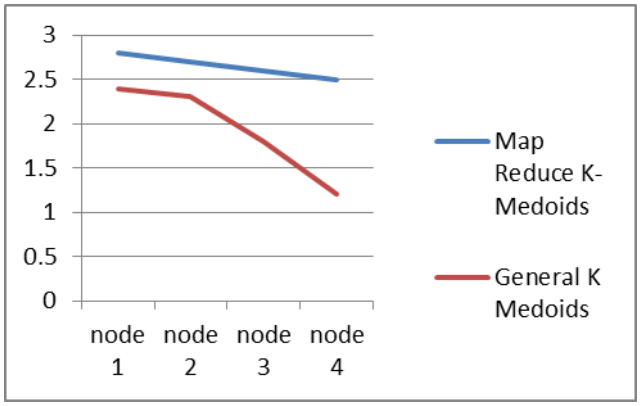
Input: *key* is the index of the cluster, *V* is the list of the partial sums from different host

Output: < *key* , *value*> pair, where the *key*' is the index of the cluster, *value*' is a string representing the new center

1. Initialize one array record the sum of value of each dimensions of the samples contained in the same cluster, e.g. the samples in the list C;
2. Initialize a counter *NUM* as 0 to record the sum of sample number in the same cluster;
3. while(C.hasNext()){
Construct the sample *instance* from C.next();
NUM += *number*;
4. }
5. Divide the entries of the array by *NUM* to get the new median;
6. Take *key* as *key*';
7. Construct *value*' as a string comprise of the *center*'s coordinates;
8. output < *key*, *value*> pair;
9. End

III. Experimental Results

In this section results has been shown of our proposed algorithm in the sense of size(X axis) and speed (Y-axis).The data size increases from node1 to node4. Our proposed algorithm performance better than existing algorithm. The difference is shown as follows.



IV. Conclusion

Many clustering algorithms have been proposed in the last past decades. These algorithms are having maximum time complexity once the data size has been increased and also there is a problem to store big amount of data. To solve these kind of problems, we propose HDFS technique to store large amount of data and fast parallel k-medoids clustering algorithm for speed up the process on the big data.

References:

[1] The Hadoop Distributed File System by Konstantin Shvachko ,Hairong Kuang,sanjay Radia and Robert Chansler.
[2]Ramissen,E.M.,Willet,P:Efficiency of Hierarchial Agglomarative clustering using the ICL Distributed Array Processor.Journal of Documentation 45(1),1-24(1989)
[3] Li, X., Fang, Z.: Parallel Clustering Algorithms. Parallel Computing 11, 275–290(1989)
[4] Olson, C.F.: Parallel Algorithms for Hierarchical Clustering. Parallel Computing21(8), 1313–1325 (1995)
[5] Borthakur, D.: The Hadoop Distributed File System: Architecture and Design(2007)
[6] Lammel, R.: Google's MapReduce Programming Model - Revisited. Science of Computer Programming 70, 1–30 (2008)
[7] Borthakur, D.: The Hadoop Distributed File System: Architecture and Design (2007)
[8] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton. MAD skills: New analysis practices for big data. In VLDB, 2009.
[9] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. SIGMOD, 2010
[10] Remzi H. Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E. Culler, Joseph M. Hellerstein, David Patterson, and Kathy Yelick. Cluster I/O with River: Making the fast case common. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems (IOPADS '99)*, pages 10–22, Atlanta, Georgia, May 1999

Author's Profile

D.Lakshmi Srinivasulu is working as an Assistant Professor in Department of Computer Science and Engineering of G. Pulla Reddy Engineering College (Autonomous), Kurnool, AP,India.. He received B.Tech degree in CSIT from JNTUH and M.Tech degrees in CSE from JNTUA. His areas of interest include Big Data, Data mining . He attended 2national conferences ,2 international conference and published 4 international journals.

A Vishnuvardhan Reddy, is working as an Assistant Professor in Department of Computer Science and Engineering of G. Pulla Reddy Engineering College (Autonomous), Kurnool, India. He received his M.E degree in Software Engineering from Jadavpur University, Kolkata, India. He received his B.Tech degree in CSE from SVU, tirupati, India. His area of research is in the design and implementation of transport layer protocol for vehicular ad-hoc networks

Dr V S Giridhar Akula has obtained PhD in Computer Science and Engineering from JNTU, Anantapur.He obtained BE and M Tech degrees in CSE. He is put up with 22 years of teaching and 2 years industry experience. Presently working as Professor and Principal at Methodist College of Engineering and Technology, Abids,Hyderabad, India. He is a life member of ISTE, IEEE,IAE,ICES,AIRCC, Member of Computer Science Teacher's Association, USA,Member of IAE. He has authored 8 text books and editorial board member for 8 national and 12 International Journals.